The ZeroAccess Auto-Clicking and Search-Hijacking Click Fraud Modules

Paul Pearce
Chris Grier
Vern Paxson
Vacha Dave
Damon McCoy
Geoffrey M. Voelker
Stefan Savage

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2013-211
http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-211.html

December 16, 2013

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE
16 DEC 2013

2. REPORT TYPE

3. DATES COVERED
00-00-2013 to 00-00-2013

4. TITLE AND SUBTITLE
The ZeroAccess Auto-Clicking and Search-Hijacking Click Fraud Modules

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of California at Berkeley,Electrical Engineering and Computer Sciences,Berkeley,CA,94720

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

ZeroAccess is a large sophisticated botnet whose modular design allows new ?modules? to be downloaded on demand. Typically each module corresponds to a particular scam used to monetize the platform. However, while the structure and behavior of the ZeroAccess platform is increasingly wellunderstood the same cannot be said about the operation of these modules. In this report, we fill in some of these gaps by analyzing the ?auto-clicking? and ?search-hijacking? modules that drive most of ZeroAccess?s revenue creation. Using a combination of code analysis and empirical measurement, we document the distinct command and control protocols used by each module, the infrastructure they use and how they operate to defraud online advertisers.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**23**	

The ZeroAccess Auto-Clicking and Search-Hijacking Click Fraud Modules (Technical Report)

Paul Pearce[†*] Chris Grier[†*] Vern Paxson[†*]
Vacha Dave[‡] Damon McCoy[◇] Geoffrey M. Voelker[‡] Stefan Savage[‡]

[†]University of California, Berkeley
[*]International Computer Science Institute
{pearce,grier,vern}@cs.berkeley.edu

[‡]University of California, San Diego
{vdave,voelker,savage}@eng.ucsd.edu

[◇]George Mason University
mccoy@cs.gmu.edu

December 5, 2013

Abstract

ZeroAccess is a large sophisticated botnet whose modular design allows new "modules" to be downloaded on demand. Typically each module corresponds to a particular scam used to monetize the platform. However, while the structure and behavior of the ZeroAccess platform is increasingly well-understood, the same cannot be said about the operation of these modules. In this report, we fill in some of these gaps by analyzing the "auto-clicking" and "search-hijacking" modules that drive most of ZeroAccess's revenue creation. Using a combination of code analysis and empirical measurement, we document the distinct command and control protocols used by each module, the infrastructure they use, and how they operate to defraud online advertisers.

1 Introduction

Botnets — large collections of malware infected computers, controlled by single entity and working together for a common goal — have become the modern platform via which mass cybercrime is waged. In particular, botnets today are central to a broad range of scams including e-mail spam, credentials theft, account abuse, denial-of-service, search engine optimization and click fraud. The structure and composition of these botnets, how they maintain command and control, and the means by which they are monetized has evolved considerably over the last decade, with each new generation offering additional resilience and flexibility.

Today one of the more sophisticated products of this evolution is *ZeroAccess*. Active in a variety of forms since 2009 [5], ZeroAccess is one of the largest botnets in operation today with over 1.9 million infected computers estimated to fill its ranks as of August 2013 [17]. ZeroAccess is further distinguished by being the best known botnet primarily monetized via *click fraud* [17], and some researchers estimate the resultant cost to advertisers at over $2.7 million USD per month [22].

It is this latter feature that motivates our report. Advertising drives much of today's Web services, generating over $20 billion in revenue in the first half of 2013 and growing at an estimated 20% per year [18]. However, this same value has also attracted criminal actors who use a variety of techniques to generate synthetic advertisement clicks to defraud advertising networks [4]. As a result, this *click fraud* accounts for as much of 10% of all advertising clicks, potentially defrauding advertisers of hundreds of millions of dollars annually, with some experts predicting the rate increasing by more than 50% per year [20].

As one of the largest click fraud botnets in existence, ZeroAccess's operations are of unique interest in understanding how mass click fraud campaigns are perpetrated. Much of ZeroAccess has been well studied, including the infection vector and the peer-to-peer (P2P) command-and-control (C&C) protocol [5, 17, 22, 23], and several reports have identified its use of click fraud [17, 22]. However, we are unaware of public work documenting the click fraud process in technical depth.

In this report we focus on documenting the click fraud behavior of the ZeroAccess botnet and the infrastructure it uses in pursuit of this goal. We take a multifaceted approach, including a combination of binary and network analysis, malware execution, and direct interaction with the botnet C&C servers. In particular, we describe how ZeroAccess uses two different "modules" to carry out distinct forms of click fraud: auto-clicking and search-hijacking.

Auto-Clicking: This ZeroAccess module automatically clicks on advertisements sent via the module's C&C. These clicks occur rapidly, unseen by the user and independent of any user interaction. Section 5 describes the behavior and infrastructure of the ZeroAccess auto-clicking module.

Search-Hijacking: This ZeroAccess module fetches ads relating to *real search queries* generated by the user on the infected machine. When the user clicks on a search result, the module intercepts the click and instead performs a separate fraudulent ad click related to that search query. It then redirects the user to an advertiser's Web site. Given the user interaction and the click's relation to the search query, such fraud may lead to advertising conversion,[1] resulting in higher revenue for the criminals. We discuss this process, and the associated mechanisms used to achieve it, in greater depth in Section 6.

The remainder of this report first explains the mechanics of Internet advertising and click fraud, the history of ZeroAccess, and our measurement methodology. We then describe the basic structure of the ZeroAccess malware distribution platform and provide a detailed description of each of its two click fraud modules. In particular we document how each click fraud module uses its own C&C network (also distinct from that used by the ZeroAccess platform) with well over a dozen server IP addresses implicated in our analysis.

2 Background

The ZeroAccess botnet exploits the Internet advertising ecosystem to profit by defrauding advertisers. In this section we frame this ecosystem, the general problem of click fraud, and the evolution of the ZeroAccess botnet.

2.1 Web Advertising

Advertising on the Web works in terms of arrangements between *advertisers*, who wish to display promotional content, and *publishers* (such as blogs, news sites, or search engines), who receive visits from users who could potentially view and respond to that content. Publishers receive payment for displaying the advertiser's content, which can consist of text, images, video, or other interactive (e.g., Flash-based or

[1] In advertising, a conversion is a click that leads to some user interaction on the advertiser's Web page. What constitutes a conversion can vary based on advertiser. Such clicks are said to be "high quality" because of the user interaction.

JavaScript-based) media. Advertisements generally include links to the advertiser's site to allow interested users to directly engage with the advertiser by clicking on the advertisement and visiting the site.

In practice, advertisers and publishers often do not deal with each other directly. Instead, each contracts with an *advertising network* that coordinates ad placement between many advertisers and publishers. In a traditional arrangement, an advertiser buys a given volume of advertising from the ad network, usually also specifying a set of *keywords* defining the context in which to show the ad. Publishers then join an advertising network and display ads provided by the network.

2.1.1 Pricing and Payments

Advertising networks price ads in one of three basic ways. For cost-per-impression pricing, the advertiser pays for each end-user *impression*; essentially, whenever a browser loads their ad as part of a Web page. Such pricing is also termed cost-per-mille (*CPM*), reflecting the common use of a thousand impressions as the usual unit of pricing. For cost-per-click (*CPC*) pricing, advertisers pay whenever a user clicks an ad, presumably reflecting potential interest on the part of the user. For cost-per-acquisition (*CPA*) pricing, advertisers pay when a user *converts*, evincing engagement with the advertiser's site such as adding an item to their shopping cart or signing up for a mailing list. CPC ads predominate today in terms of ad revenue [18].

Another consideration of relevance to the mechanics of advertising and pricing concerns whether the publisher offers *search* or *contextual* placement for the ads. To match ads to relevant publisher sites, advertisers usually provide associated keywords. Search ads are placed by publishers (such as search engines) in response to specific user searches on the publisher's site, while contextual ads reflect matches to words present on a publisher's site (such as a blog post). Since users have more explicit intent when searching, search ads typically lead to higher conversions and thus cost more.

With cost-per-click (CPC), publishers usually take a 70% cut of the cost of the ad, while the other 30% goes to the ad network. CPC can be quite lucrative for publishers, with prices for certain keywords as high as $50 per click [9].

Publishers may have syndication arrangements with other, smaller publishers (for example, a large search engine syndicating to a smaller search engine), where the publisher may choose to provide ads from the ad network to the smaller publisher. When such a syndicated ad results in a click, the syndicating publisher takes a cut of the profit. Syndication chains can be arbitrarily long; e.g., publisher A may sub-syndicate to publisher B, who may in turn sub-syndicate to publisher C, and so on. Typically every publisher along the chain earns money for a click, the amount depending on the type of syndication agreement (flat-rate vs. a fraction of the profit).

2.1.2 Anatomy of a click

When an advertiser decides to advertise with an ad network, the advertiser provides the ad content to display to users along with a *landing* URL on the advertiser's site to which clicks on the ad should take users. When a publisher decides to use a particular ad network to populate some part of its Web page, the network provides the publisher with a code snippet to fetch ads from the network's servers. This snippet can be embedded in the Web page (typically using JavaScript), or it can fetch ads in the background that it then formats before displaying to the user.

Figure 1 diagrams an example interaction between a user visiting a publisher site, an ad network, and an advertiser. When a user receives a requested publisher page (*Steps 1–2*), the JavaScript snippet embedded in the page contacts the ad servers to request ads (*Step 3*), identifying the publisher that made the request. In response, the ad server decides which ads to provide, and logs the request, an **ad impression** (*Step 4*). Next,

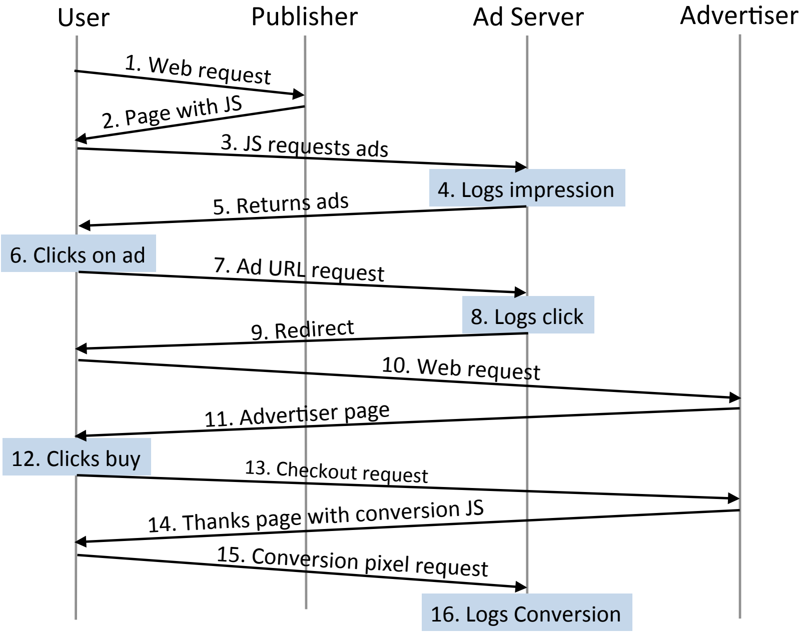

Figure 1: Typical anatomy of an ad click, showing the various HTTP requests associated with a user clicking on an advertisement, leading them to an advertiser's *landing page*, and from there possibly to additional interactions.

the ad server returns to the user's browser the ad content along with a unique ad URL (*Step 5*). The ad URL contains an identifier used for linking it to the advertiser.

If the user chooses to click on the ad (*Step 6*), the browser makes an HTTP request for fetching the ad URL from the ad server. At this point, the ad server logs an **ad click** (*Step 7*) and, if using the predominant *CPC* pricing, charges the advertiser associated with that ad. After logging the click, the server redirects the user's browser to the advertiser's site, typically using an HTTP redirect response code (*Steps 9–11*), though other mechanisms are possible.

After landing at the advertiser Web site, the user may choose to perform an action desirable to the advertiser (*Step 12*), a **conversion**. The advertiser decides what constitutes a conversion, in general choosing actions that represent a tangible return on investment. Most ad networks provide optional conversion-tracking by having the advertiser embed a single-pixel image hosted by the ad network on the desirable page, which enables linking together the click and the conversion events based on a cookie value (*Step 16)*.

2.2 Click Fraud

In the context of CPC ads, click fraud is the practice of fraudulently generating clicks on CPC ads without any intention of fruitfully interacting with the advertiser's site. As a result, advertisers lose money, receiving no return on their investment.

There are two primary motivations for click fraud. First, a malicious advertiser can employ click fraud targeting a competitor's ad to deplete their advertising budget [21]. A stronger motivation, however, lies with publishers, who directly profit from ads clicked on their site. Ad networks also profit from click fraud, though reputable ad networks that want to maintain long term relationships with advertisers will presumably attempt to identify and weed out click fraud activity. Note that it is hard for an ad network to prove that a particular click was fraudulent because it is equivalent to guessing the intent of the user behind the click.

An attacker can perform click fraud in various ways. Simple approaches involve hiring people to click on ads (termed *click fraud farms*) [6] or running stand-alone scripts that repeatedly retrieve the URLs associated with ads, simulating user clicks (*stand-alone click fraud bots*) [13]. As ad networks' defenses have evolved, so have the means of click fraud. The more complex approaches includes *search engine hijacking* [19], where a malicious in-browser plugin replaces ad links in results returned for user searches by other ads, confusing a user into clicking; and the use of *click fraud botnets* [16, 22], i.e., groups of malware-infected hosts that fetch and click ads unbeknownst to the user.

Of these, botnets can pose a very difficult case for an ad network to detect due to the large geographic distribution from which the fraudulent clicks appear, which combined with a low click fraud intensity can make the activity seemingly indistinguishable from genuine user traffic [3]. One tactic that many ad networks employ, *smart pricing* [7], discounts clicks from publishers that subsequently do not lead to conversions, based on the assumption that many forms of click fraud can force clicks but have difficulty producing conversions.

2.3 ZeroAccess

ZeroAccess is a complex botnet that has undergone several stages of evolution, which we recount here. Although first described solely as a "rootkit," ZeroAccess has developed into a vast *peer-to-peer* (P2P) botnet and malware delivery platform.

2.3.1 Early Life, 2009–2011

Initial reports of the "ZeroAccess rootkit" date to 2009 [5]. In 2010, the InfoSec Institute's detailed analysis described it as a "platform to deliver malicious software" [2]. At this stage, the main malware delivered using ZeroAccess was "FakeAV",[2] with an estimated 250,000 computers infected.

2.3.2 First generation P2P Botnet, 2011–Present

In May 2011 a radically new version of ZeroAccess emerged [17]. In this iteration, ZeroAccess retained its kernel-mode rootkit components, but changed both its communication and monetization strategies. This version spread itself via exploit packs (e.g., BlackHole [8]) and social engineering [14, 23].

The defining feature of this iteration of the botnet was the introduction of a decentralized, TCP-based P2P communication protocol. The protocol used cryptography and obfuscation as well as other common P2P features such as "super nodes" that served to orchestrate large portions of the network's activity. The network allowed the botmasters to maintain decentralized control while relaying commands and payloads to infected computers worldwide [22].

The P2P protocol included cryptographic signing of malicious payloads, which hardened the botnet against attempted hijacking by preventing untrustworthy peers in the botnet from successfully delivering payloads other than those cryptographically signed by the actual botmaster [22].

In addition, the monetization strategy changed with this generation. ZeroAccess moved away from FakeAV payloads and instead began distributing Bitcoin miners and click fraud modules.[3] From a technical perspective, the primary click fraud malware used in this era operated in the indiscriminate "auto-clicking" fashion we describe in Section 5.

Alongside the click fraud and Bitcoin payloads, ZeroAccess itself was also sold as a service on underground forums [17], enabling cyber-criminals to use the ZeroAccess rootkit to distribute their own malicious payloads.

[2] FakeAV is malware that claims to be anti-virus software to extort users into paying money to remove fictitious infections.

[3] ZeroAccess's shift away from FakeAV occurred just before a major takedown that resulted in the closure of most FakeAV programs [11].

MD5	*Last Obtained*
Auto-Clicking module	
51ba6261e44c60b2f891fabfaa47d0ad	Nov. 22, 2013
Search-Hijacking module	
7128a957f5c9c9a69385f5332ca6338c	Nov. 22, 2013
3aec103d38c7520229e18af260c5a00d	Sep. 26, 2013
36616e8f309b35f8e090068690272239	June 14, 2013
8fa08c59e4d205e514f8a978678ba798	May 30, 2013

Table 1: Auto-Clicking fraud and Search-Hijacking module executables used in the analysis.

This iteration of the botnet also saw an increase in botnet population. At the height of infections in early 2012, estimates placed the botnet population at over 500,000 [15]. Despite the age of the botnet and its subsequent evolution, as of August 2013 there were still over 30,000 computers infected with this generation [17].

2.3.3 Second generation P2P Botnet, 2012–Present

In July 2012, ZeroAccess evolved into the form predominant as of November 2013. According to Symantec, by August 2013, this generation had an estimated population of over 1.9 million computers [17]. This iteration included several changes to the malware structure, the protocol, and the payloads. The most distinguishing change to ZeroAccess in this era was a move away from the kernel-mode rootkit component, with all of its functionality now replicated in user-space [17]. Other changes include a move to UDP from TCP for the P2P protocol (likely to improve network performance), and minor changes in the protocol itself.

The monetization strategy also evolved. This version saw the introduction and massive distribution of a new click fraud payload performing *search-hijacking* , which we discuss in Section 6.

3 Methodology

In this section we describe our malware execution environment, manual analysis techniques, and the ZeroAccess modules we examine.

3.1 Collecting Module Sample Executables

Table 1 lists the modules used in our analysis of the click fraud modules. We obtained our samples of ZeroAccess by searching malware repositories for traffic patterns consistent with ZeroAccess command-and-control (C&C) behavior and executing each binary in our execution environment (Section 3.2). We have identified thousands of binaries with ZeroAccess C&C behavior found in October and November, 2013.

During execution, ZeroAccess retrieves the auto-clicking and search-hijacking modules for execution. While ZeroAccess transfers modules in an encrypted form, using reverse engineered decryption routines [22] we built a tool that automatically extracts modules from network traces.

3.2 Monitored Execution Environment

We execute each binary in a virtualized environment provided by the GQ honeyfarm [12], which supports monitoring malware execution while providing a flexible network containment policy. We use Windows XP Service Pack 3 for all executions. The system can process thousands of binaries per day.

In our experiments the execution environment allows ZeroAccess C&C P2P (UDP) traffic. For all executions we forward HTTP traffic to the intended destination and redirect all other non-C&C TCP traffic to internal sinks. For DNS, our service can answer all queries, even requests without a valid answer or directed at external DNS servers. This feature ensures that domain takedowns during our analysis have limited impact on malware execution. The configurable nature of the DNS server behavior enables us to test ZeroAccess samples with and without DNS resolution. For all other protocol types we provide a sink that will accept packets but does not respond.

In addition to network monitoring, our system collects operating system events, including process creation, file modifications, and registry changes.

3.3 Binary Analysis

For static binary analysis we use IDA Pro 6.4 with the Hexrays decompiler.[4] ZeroAccess distributes modules as standard Windows DLLs, a file format natively supported by IDA such that it can disassemble and decompile the modules with Hexrays. We use static binary analysis to obtain the encryption (and decryption) algorithms, domains, and other C&C protocol information for the ZeroAccess modules.

3.4 Milking

A *milker* is a program that speaks a particular botnet's C&C protocol and mimics the communications of that malware. Through the use of a milker, we can query information and commands at a much larger scale, with finer granularity, and across more diverse geographic regions, than with traditional malware executions. This technique also allows us to probe specific protocol behaviors in a way that directly executing the malware might not manifest, and to obtain C&C commands without potentially dangerous malware side effects.

For this work we created a milker for the ZeroAccess search-hijacking module's C&C protocol and used it to interact with the module's C&C servers. The milker queries the C&C server and retrieves a list of ads to click on, then simulates a click on one of the results using a headless Web browser. The browser follows redirects, executes JavaScript, and in general is designed to perform similarly to a victim's browser. We ran our milker over several days and describe some of the data gathered in detail in Section 6.

4 The ZeroAccess Platform

In this section we describe the base ZeroAccess platform: the botnet software responsible for coordinating communication among very large numbers (millions) of infected computers around the world.

4.1 Infection

The first step in a ZeroAccess victim's lifecycle is becoming infected with ZeroAccess. Like many other malware families, ZeroAccess is distributed in a variety of ways, such as drive-by download, social engineering, and pirated software [14]. Each distribution vector results in the installation of software that participates in the ZeroAccess C&C.

An example of this process we have observed begins when a victim browses the Web and inadvertently visits a compromised Web site hosting an exploit kit. The exploit kit detects the victim's browser version and delivers an exploit payload. The payload has two functions: 1) exploit a browser vulnerability, and 2) deliver malware. Upon successful browser exploitation, the payload downloads and executes a ZeroAccess binary. Once executed, ZeroAccess has control of the victim's computer and begins to communicate using the P2P C&C protocol.

[4] https://www.hex-rays.com/products/ida/index.shtml

4.2 Command and Control

The ZeroAccess platform uses a P2P protocol for its C&C, with the primary function of distributing modules and performing updates. The P2P protocol, described in greater detail in other reports [17, 22], supports the promotion of a member to a super node. As described by Neville *et al.*, super nodes store ZeroAccess modules and provide them to other nodes upon request [17]. In addition to distributing modules to newly infected hosts, super nodes also host new versions of modules when updated. It is important to note that aside from distributing the click fraud modules, the P2P C&C protocol does not play a role in the execution of click fraud.

Once a victim has been infected with ZeroAccess, it begins by bootstrapping the P2P protocol using a peer list embedded in the binary [17]. The P2P protocol discovers new peers, updates the peer list, and adds itself to the peer list for new nodes to contact. Once the newly infected machine joins the ZeroAccess P2P network, it begins to download modules as instructed by other peers in the network. A super node hosts the modules, and the malware issues a download request to fetch and then execute the module. This process occurs shortly after infection and results in a click fraud module download. Once that module executes, the victim's computer begins to carry out click fraud using a separate C&C protocol as described in Sections 5 and 6. When an update to the click fraud modules becomes available on the P2P network, the victim learns of the update from one of its peers and contacts a super node to retrieve the latest version.

5 The Auto-Clicking Module

The ZeroAccess auto-clicking module performs click fraud by simulating normal Web browser behavior of a user clicking on a Web advertisement. This activity requires no user participation, and is not visible to the user.[5] We present the behavorial analysis of this module, treating the module itself as a black box and observing its contained execution. We observed it performing about one click every two minutes. To evade detection, these clicks were spread across multiple ad networks.

5.1 Behavior

The function of the auto-clicking module is to simulate a user "click" on an advertisement. Figure 2 shows the operation of the module. The module begins by contacting one of its command-and-control (CF-C&C) servers with a request for click fraud jobs (Step ❶). The C&C server returns a scrambled payload containing a list of "click" jobs. Each job is identified by a host name, a first hop URL and the HTTP `Referer`[6] URL. The module then issues an HTTP request to another CF-C&C server, setting the `Host` header value as specified by the job. This server then redirects the request via an HTTP 303 redirect to a URL, the same URL as in the job (Step ❷). This forms the first hop in the redirection chain. The module then retrieves the URL to which it is redirected, setting the `Referer` header as given in the job (Step ❸). The redirect chain continues normally from this point on, and after a series of redirects the bot fetches the ad URL, at which time the advertiser gets charged (Step ❹).

There is no visible activity in the foreground, so it is difficult for a user to detect that their computer is performing click fraud in the background. This same process repeats multiple times.

[5] Some users, however, may be alerted to the presence of this module by the increased network activity; in one instance, we observed about 50 MB of network traffic per hour.

[6] An HTTP `Referer` header provides the server with the URL of the referring page, that is, the page that purportedy contained the URL being requested.

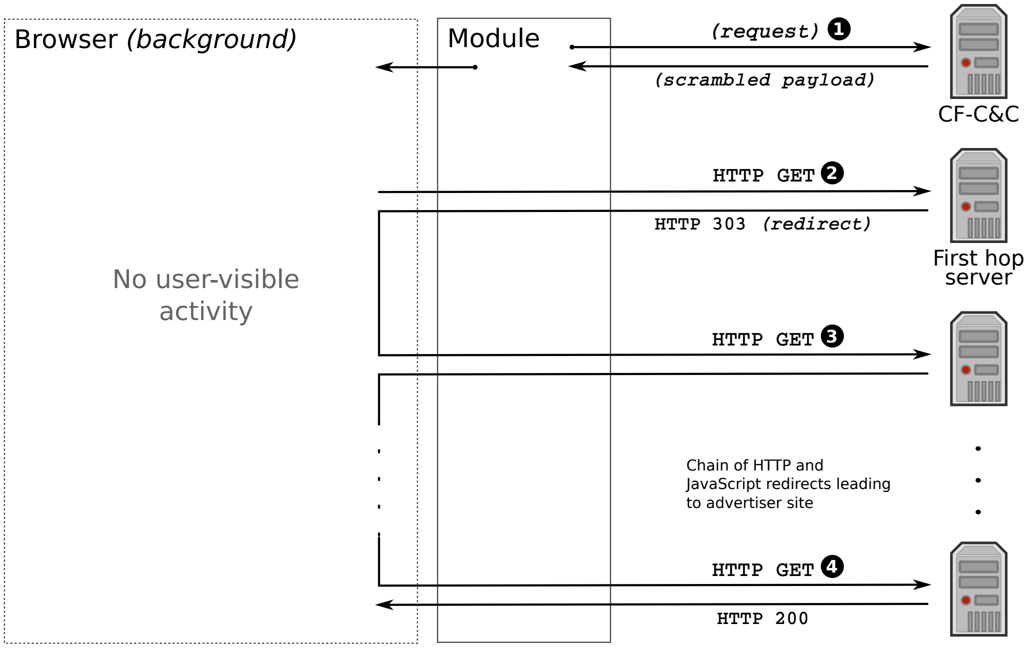

Figure 2: Behavior of the auto-clicking module. The module begins by retrieving a list of "click" jobs from its C&C server (Step ❶). For each job, it uses the system browser to retrieve the URL (Step ❷) and follows HTTP and JavaScript redirects (Steps ❸ and ❹).

5.2 Command and Control

This section describes how the bot communicates with the CF-C&C servers to fetch click jobs. Table 2 lists the IP addresses of the CF-C&C servers that we observed the bot contacting for fetching commands. Note that these change over time.

The bot contacts one of the CF-C&C servers over TCP port 12757, and sends an obfuscated message (message string XORed by 0x72) identifying the browser User Agent string. In response, the CF-C&C server sends a response (also obfuscated) that contains the following: a domain name, list of first hop URLs to be contacted and a set of `Referer` headers to be set.

After receiving the first hop URLs, the auto-clicking module does not fetch the first hop URLs directly. Instead, it first contacts one of the other CF-C&C servers over HTTP port 80, with the `Host` header set to the domain name provided earlier. Presumably this is an authentication mechanism; earlier versions of the

IP Address	*Observed Date*	*Location*
94.242.195.162	21 Nov 2013	Luxembourg
94.242.195.163	21 Nov 2013	Luxembourg
94.242.195.164	21 Nov 2013	Luxembourg
81.17.18.18	21 Nov 2013	Switzerland
81.17.26.189	21 Nov 2013	Switzerland
46.19.137.19	21 Nov 2013	Switzerland

Table 2: IP addresses of C&C servers observed for the auto-clicking module in Table 1. The *Obeserved Date* gives the date on which we observed communication between the module and these servers. Location is based on MAX-MIND [1] GeoIP service.

Hop	*URL*	*Status Code*	*Notes*
1	`http://46.19.137.19 ...`	303	Host name set to cvmrpznw.cm
2	`http://1556987547.traffiliator.com ...`	302	Referer spoofed
3	`http://unlimiclick.com/bd ...`	200	
4	`http://ads.clicksor.cn ...`	200	
5	`http://poomedia.com/ad ...`	200	Loaded in iframe
6	`http://us.ad2mi.com ...`	302	Loaded in 1x1 pixel iframe
7	`http://searchists.com/search ...`	200	JavaScript redirect
8	`http://searchists.com/click/ ...`	302	
9	`http://click.local.com ...`	302	
10	`http://1389.r.msn.com ...`	302	Ad URL fetch, advertiser charged
11	Advertiser	200	

Table 3: Example redirection chain corresponding to an auto-clicking module "click" from November 21, 2013.

module exhibited similar behavior of not fetching the URLs directly [22]. In response, the second CF-C&C server sends an HTTP 303 response code, redirecting the browser to one of the URLs in the list. At this point, the auto-clicking module inserts a supplied `Referer` header and a click chain begins. After a series of redirects, the ad URL gets fetched, resulting in the advertiser getting defrauded.

5.3 Example redirect chain

Table 3 shows a redirect chain generated by the auto-clicking module, starting from when the bot contacts the C&C server to authenticate itself by setting the `Host` header. In response, the CF-C&C server at `46.19.137.19` redirects the bot to `1556987547.traffiliator.com`. This URL was present in the original click fraud job list, and the bot now inserts the corresponding referrer before fetching the URL, which eventually causes a banner to be fetched in an iframe from `poomedia.com`. In addition to the banner, `poomedia.com` also populates the banner iframe with another 1x1 pixel iframe. This second iframe is loaded with ad URLs and JavaScript code that automatically fetches one of the links at random. This step results in an ad click, which is eventually redirected through other publishers to the ad network, and eventually to an advertiser.

5.4 Entities

Depending on the syndication arrangement between different parties in a redirection chain, all of them stand to gain from a fraudulent click that an advertiser pays for, and thus any one of the publishers may be working with the botnet. Over time, from our observations and others [22], different versions of this module have been seen to defraud all major CPC ad networks, including AdCenter, AdWords, 7Search, affinity, and adsimilate. We speculate that this module evades ad network detection by spreading click fraud across a large number of ad networks to hide the high volume of click fraud performed.

Since this click fraud is invisible to the end user (unlike the search-hijacking click fraud module), the user is unlikely to convert. Given that, the use of *smart pricing* (cf. Section 2.2) should in theory discount such malware-driven clicks. However, because of the large number of hops in the syndication chain and the JavaScript redirects that hide the true length or source of the origin of the traffic, it becomes extremely difficult for an ad network to identify that the traffic is being driven by a malware source, as the click fraud traffic mixes in with other legitimate (and converting) traffic from its known syndicators, thus undermining the use of smart-pricing.

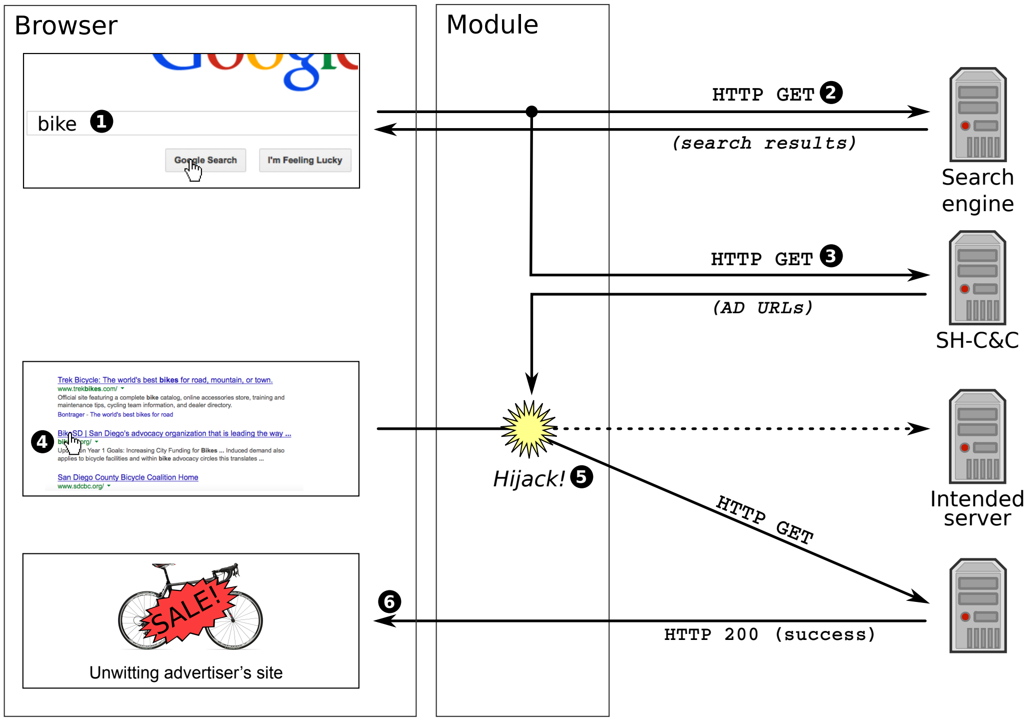

Figure 3: Behavior of the search-hijacking module. **Step** ❶ : A user enters a term into a search engine. In this example, the user searches for "bike". **Step** ❷ : The user's browser performs an `HTTP GET` for that term. In response, the user is presented with the unaltered search result from the search provider. **Step** ❸ : In parallel to the user search, ZeroAccess sends the search term ("bike") to the ZeroAccess SH-C&C server. **Step** ❹ : The user clicks on a search result on the unaltered results page. **Step** ❺ : ZeroAccess intercepts the click. Rather than going to the intended click destination, the user is sent to one of the ad URLs supplied by the ZeroAccess SH-C&C in Step ❸. **Step** ❻ : The user's browser displays the result of the ad click, an advertising landing page related to their original query ("bike").

6 The Search-Hijacking Module

The search-hijacking module interposes on a user's normal interaction with various search engines in order to redirect the user to an advertisement that generates the botmaster revenue. Such *search-hijacking* represents a more sophisticated type of click fraud. Whereas the auto-clicking module simulates a real user, the search-hijacking sends a real user to the advertiser. Because the advertiser's site is relevant to the user's search query, the user may in fact interact with the advertiser's site and trigger a conversion, as described in Section 2. We describe the module's search-hijacking behavior in more detail next, and then report on our analysis of this module.

6.1 Behavior

Once loaded, the module monitors the the interaction between the user on an infected PC and the browser, waiting for the user to issue a search query to a search engine (Step ❶ in Figure 3). We have confirmed that the module recognizes and hijacks Web searches performed using Google, Bing, Yahoo, Ask, and ICQ Search. The module captures the query terms, while allowing the query to go through to the intended search engine (Step ❷). At the same time, the query terms are sent to the search-hijacking module's C&C (SH-C&C) server to retrieve a list of ad URLs to be used for later hijacking (Step ❸). When the user clicks on a search or ad result (Step ❹), the normal click is *hijacked* and the intended URL is replaced with the replacement ad URL retrieved from the SH-C&C server (Step ❺). The browser then fetches and renders the

Hop	*URL*	*Status Code*
1	`217.23.3.223/...`	302
2	`http://feed.hype-ads.com/...`	302
3	`http://search.freshcouponcode.com/search.php...`	200
4	`http://c.freshcouponcode.com/redrct.php...`	200
5	`http://c.freshcouponcode.com/click.php...`	301
6	`http://nn.xdirectx.com/clicklink.php...`	302
7	`http://2478799.r.msn.com/...`	302
8	Advertiser	200

Table 4: Example of a redirect chain corresponding to a "click" issued by the search-hijacking module on November 14, 2013. Non-final hops with 200-level status codes trigger Javascript or Flash-type redirects.

replacement URL instead of the intended search result URL (Step ❻). Although not shown in the figure, retrieving the replacement URL may involve a chain of HTTP and JavaScript redirects. Table 4 gives an example redirect chain of a click issued by the module.

The replacement and redirection process operates invisibly to the user. An unsuspecting user will believe that the resulting page corresponds to the search result or ad on which the user clicked on the search result page. It is important to note that neither the advertiser nor the ad network used in the hijacked click may be aware that search-hijacking took place. From their point of view, a hijacked user appears no different from a user arriving via normal search syndication.

Unlike traditional click fraud, such search-hijacking actually delivers a user to the advertiser. Such a user may interact with the advertiser's site and even convert, as described in Section 2.1.2. Because some fraction of the users will convert, smart pricing may treat traffic from the affiliate engaged in search-hijacking as legitimate and pay for each click.

6.2 Command and Control

The search-hijacking module's command-and-control (SH-C&C) protocol uses HTTP with a hard-coded set of server addresses. We now detail the different elements of this protocol.

6.2.1 Commands

The primary purpose of the SH-C&C channel is to retrieve a list of replacement URLs to which the user will be redirected when clicking on a query result. When the user performs a search engine query, the module makes an HTTP GET request to a SH-C&C server (Step ❸ in Figure 3). The HTTP GET request string is formed as shown in Figure 4. The user search terms, together with additional parameters is first formatted using standard URL parameter encoding, using the printf-style format string:

```
v=5.4&id=%08x&aid=%u&sid=%u&q=%.*s&eng=%.*s&os=%s&br=%S&s=%u
```

The parameter string is then encoded using Base64 encoding and padded with 13 randomly-generated[7] characters at the front and 10 similarly-generated characters at the end. The length of the padding is such that a trivial decoding of the entire string does not reveal the contents of the message. The resulting string is then sent to the SH-C&C server in the HTTP GET request.

[7] The malware generates the padding characters using the Windows random number API.

IP Address	v=	*Pseudo-Domain*	*Purpose*
195.3.145.108	5.4	dclixvfpttrlcnindvrnyeic.com	Search request
	5.4	evtrdtikvzwpscvrxpr.com	
	5.3	atenrqqtfrzozqrqbdzwkxzyuc.com	
83.133.120.186	5.4	gozapinmagbclxbwin.com	Search request
	5.4	nbqkgysciuuhadgpjfquvpu.com	
	5.3	cjelaglawfoyidgyapv.com	
83.133.120.187	5.4	jpciukjdkqxgreoikpgya.com	Search request
	5.4	qhdsxosxtvmhurwezsipzq.com	
	5.3	omakfdwkhrpqudxvapy.com †	
217.23.3.225	5.4	hzhrjmeeczcgxodmqyz.com	Search request
	5.4	fnyxzjeqxzdpeocarhljdmyjk.com	
	5.3	sqdfmslznztfozshtidmigmsbh.com †	
217.23.3.242	5.4	vdlhxlmqhfafeovqohwrbaskrh.com	Search request
	5.4	nmfvaofnginwocnidecxnpcs.com	
	5.3	euuqddlxgrnxlrjjbhytukpz.com †	
188.40.114.195	2.1	qvhobsbzhzhdhenvzbs.com	Click confirmation
188.40.114.228 *	2.1	mbbcmyjwgypdcujuuvrlt.com	Click confirmation
	2.0	wuyigrpdappakoahb9.com	
217.23.9.247	6.1	vzsjfnjwchfqrvylhdhxa.com	Flash player identification
	5.8	vjlvchretllifcsgynuq.com	
83.133.124.191 *	5.6	chvhcncpqttfpcibtmetg.com	Flash player identification
178.239.55.170	1.2	jgvkfxhkhbbjoxggsve.com	Unknown / JavaScript injection
83.133.120.16	1.2	xlotxdxtorwfmvuzfuvtspel.com	Unknown / JavaScript injection
	1.1	mkvrpknidkurcrftiqsfjqdxbn.com	
83.133.124.191	—	ezcfogjitbqwnornezx.com	Fallback
	—	rwdtklvrqnffdqkyuugfklip.com	
	—	uinrpbrfrnqggtorjdpqg.com	
188.40.114.228	—	jzlevndwetzyfryruytkzkb.com	Fallback
	—	glzhbnbxqtjoasaeyftwdmhzjd.com	
	—	kttvkzpwufmrditdojlgytxyb.com	
46.249.59.47	—	loanxohaktcocrovagkaa.com	Fallback
	—	mxyawkwuwxdhuaidissclggy.com	
	—	erspiwscuqslhjflgbbgcfbc.com	
46.249.59.48	—	spujplpdupiwbghiedhqeja.com	Fallback
	—	xttfdqrsvlkvmtewgiqolttqi.com	
217.23.9.140	—	dxgplrlsljdjhqzqajkcau.com	Fallback

Table 5: Pseudo-domains and IP addresses extracted from the search-hijacking module via malware executions and reverse engineering. The v= column shows the value of the v argument used in requests. The *Purpose* column lists the class of commands sent to the SH-C&C server. Communication attempts to (and DNS requests for) *Fallback* IP addresses occur when the malware is unable to establish communication with a pseudo-domain selected for another function. When this occurs, the original SH-C&C message is sent to the fallback IP address. In this case, a pseudo-domain corresponding to the fallback address does *not* appear in the HOST field; instead, the original pseudo-domain appears. Pseudo-domains labeled with † were discovered via reverse engineering, but not verified in observations of network requests, presumably due to limited executions. IP addresses labeled with * reflect addresses that were unexpected given the associated pseudo-domains. This anomalous behavior occurred in a very small number of executions, perhaps due to some kind of bug, or related to the fallback domains. We inferred the IP addresses associated with predicted but not observed pseudo-domains via the de-obfuscation algorithm.

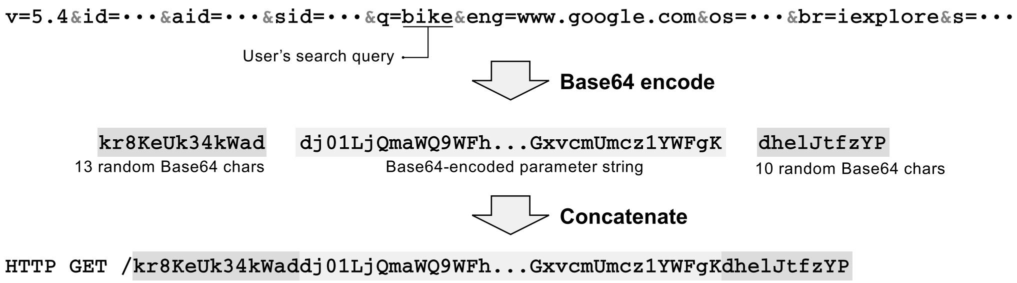
v=5.4&id=•••&aid=•••&sid=•••&q=bike&eng=www.google.com&os=•••&br=iexplore&s=•••
User's search query
Base64 encode
kr8KeUk34kWad
13 random Base64 chars
dj01LjQmaWQ9WFh...GxvcmUmcz1YWFgK
Base64-encoded parameter string
dhelJtfzYP
10 random Base64 chars
Concatenate
HTTP GET /kr8KeUk34kWaddj01LjQmaWQ9WFh...GxvcmUmcz1YWFgKdhelJtfzYP

Figure 4: Encoding of a ZeroAccess search-hijacking module search request. When the user issues a search query, the module requests a list of URLs to which the user should be redirected. The user's query and other module parameters are combined using the standard URL parameter encoding scheme. The resulting string is then Base64-encoded and padded by prepending 13 and appending 10 apparently random Base64 encoding characters. The resulting string is then used to form the HTTP GET request to the SH-C&C server. (Values denoted "···" have been truncated for space in this example.)

IP Address	*Location*
217.23.3.223	Netherlands
83.133.127.85	Germany

Table 6: IP addresses of the first hop servers in the ad click redirection chain for the search-hijacking module.

6.2.2 Primary Rendezvous

Hardcoded into each version of the ZeroAccess search-hijacking module is a list of `.com` domains of the form shown in Table 5. However, these domains are not resolved in the usual way using the Domain Name System. Instead, each domain encodes an IP address directly. To make the distinction clear, we call these *pseudo*-domain names. To obtain the IP address of a command-and-control server, the module decodes one of the pseudo-domain names to an IP address using the algorithm given in Appendix A.2, which we extracted from the module binary. In addition to decoding to an IP address, the domain name may have been used for authentication as described in the next section.

Table 5 lists all pseudo-domain names and their associated IP addresses and domain names associated with the search-hijacking module that we observed. In addition, Table 6 lists the IP addresses of the first hop servers in the search-hijacking ad click redirection chain.

6.2.3 Authentication

Normally, an HTTP interaction progresses as follows: (1) the browser resolves the domain name in the URL (e.g., `www.google.com`) to an IP address; (2) the browser connects to the Web server at the given address; (3) in its request, the browser sends a `Host` header specifying the domain name. Instead, the bot client skips the first step and from the pseudo-domain directly extracts the associated IP address encoded in the name. It then connects to the Web server and still includes a `Host` header with the pseudo-domain, even though it never resolved that name, and in fact could not since the name is unregistered in some cases.

During our early exploration of the botnet, we observed that manipulating or removing the `Host` header resulted in the SH-C&C protocol responding to messages with errors. After subsequent updates to the SH-C&C protocol, however, we were unable to reproduce this behavior.

This behavior, when active, could reflect usage of the domain name as way to authenticate legitimate bot clients to the SH-C&C server. No normal Web browser can reach the server via the domain name since it is not registered; and presumably no scanner trying to find Web servers will know which domain name to include in the `Host` header to look like a bot client.

6.2.4 Encryption

In response to a SH-C&C message, the server sends back an HTTP octet-stream of encrypted ciphertext. Appendix A.1 contains details on the decryption algorithm. (We obtained the algorithm via static reverse engineering.) The response to search result C&C requests, once decrypted, provides a list of 0-or-more replacement ad URLs. These URLs are used in Step ❺ of the hijacking process.

The target for each ad URL is a first hop ad server (Table 6), which when visited will begin a 302-redirect chain. Along with this ad click URL is another URL to be used as a forged Referer field in the subsequent ad fetch.

6.2.5 Rate Limiting

During our interaction with the ZeroAccess SH-C&C we observed advertisement click rate limiting. When we performed frequent searches from a particular IP address, the SH-C&C initially returned a large (more than 5) number of ads per query. The more we interacted with the advertisements, though, the fewer ads were returned from subsequent servers. Specifically, we observed rate limiting across the following dimensions independently: source IP address, search term, and affiliate ID.

Rate limiting based on IP address or affiliate ID may imply the botnet attempting to limit the amount of fraud performed by a particular entity, in order to avoid detection. Rate limiting based on search term may reflect a limited supply of relevant ads for a particular term.

6.2.6 Secondary Rendezvous

Prior to establishing a connection to an IP addresses derived from an obfuscated SH-C&C domain, the ZeroAccess malware performs a DNS request (an `A-Record`) for the domain. This DNS request is generated by the ZeroAccess malware, and does not use any of the traditional Windows API's for resolving domains. The request always has DNS transaction ID `0x3333` and is always sent to Google's DNS server at `8.8.8.8`.

Through both static reverse engineering and live malware executions, we have been unable to ascertain the purpose for this DNS activity. When a new version of the search-hijacking module is released, the domains associated with that module are generally not registered. Throughout the lifetime of the module various domains will become registered, sometimes by security researchers. However, the behavior of the ZeroAccess malware does not appear to be affected by the response to these DNS requests. The malware never uses the IP addresses returned by these queries, and its execution continues independent of the resolution status of the domain.

Other malware families have used similar functionality as a secondary rendezvous utilized to regain control of the botnet in the event of a takedown [10]. Although we do not believe the current ZeroAccess versions have such behavior, we are unable to definitively rule out such behavior.

6.2.7 Additional Functionality

In additional to the primary SH-C&C message that sends search terms and receives replacement ad URLs, there are three other distinct types of SH-C&C communication. The four SH-C&C messages have the same behavior with respect to how messages are formatted, obfuscated, and encrypted. These messages correspond with the *Purpose* categories in Table 5. Each of the four message types use the same primary rendezvous technique, although each pull from a distinct set of domain names.

The three additional SH-C&C messages have the following syntax:

```
v=1.2&id=%u&aid=%u&sid=%u&os=%s
v=6.1&id=%08x&aid=%u&sid=%u&os=%s&fp=%s&ad=%u
v=2.1&id=%08x&aid=%u&sid=%u&kw=%s&url=%s&ref=%s&os=%s
```

Click confirmation: After a user's click is hijacked, the malware sends a message of type `v=2.1` to a SH-C&C server, reporting the click URL as the URL parameter. In response to this message the SH-C&C server may direct the malware to perform additional clicks.

Flash player identification: Messages of type `v=6.1` report the user's Flash Player version to SH-C&C servers. The version is relayed via the `fp` parameter.

Unknown / JavaScript injection: The intended purpose of type `v=1.2` messages is unknown. In practice these messages occur far less frequently than the other types of communication. In response to this message from the malware, the SH-C&C will occasionally respond with ad network JavaScript, which we suspect is then injected into webpages viewed by the user.

6.3 Module History

Between May 2013 and November 2013 we have observed two distinct versions of the search-hijacking module (independent of changes to the included pseudo-domains). Initially (May and part of June), the `v` parameter observed in search request messages was `5.3`. During this time the response to search queries was a list of plaintext advertisements. In June, the value of the `v` parameter changed to `5.4`, and the response to search queries took on the encrypted form described above.

The `v` identifier for the other three categories of commands has also changed over time, as shown in Table 5.

We have also examined samples that had different pseudo-domain names hardcoded into them. Despite including different pseudo-domain names, the names generally decode to the same set of IP addresses.

6.4 Advertising Networks

Over time, from our observations and others [22], this module has been seen to defraud a large number of ad networks, including 7search, Affinity, Domain Development Corporation and Hoist Media. Some of these ad networks overlap with those seen defrauded by the click fraud module, but some we have only seen defrauded by the search-hijacking module.

7 Conclusion

In this report we have described two ZeroAccess modules, the *auto-clicking* module that performs traditional click fraud by simulating user clicks on advertisements, and a more recent *search-hijacking* module, which intercedes upon user clicks on Web search results, instead sending the user to an advertisement related to the search. In both cases, the botmaster stands to earn a commission from the click. We documented technical specifics for both forms in detail, including key infrastructure components (domain names and IP addresses corresponding to C&C servers) we have discovered via reverse-engineering of the modules and by observation of the malware's live execution.

Acknowledgements

We are grateful to Nick Weaver for his expert assistance and support with DNS measurement. This work was supported in part by the Office of Naval Research MURI grant N000140911081, by National Science Foundation grants NSF-1237264, NSF-1237265, NSF-1237076 and CNS-0831535, and by generous research, operational and/or in-kind support from the UCSD Center for Networked Systems (CNS).

References

[1] GeoIP databases and web serives. http://www.maxmind.com/en/geolocation_landing.

[2] G. Bonfa. Step-by-Step Reverse Engineering Malware: ZeroAccess / Max++ / Smiscer Crimeware Rootkit. http://resources.infosecinstitute.com/step-by-step-tutorial-on-reverse-engineering-malware-the-zeroaccessmaxsmiscer-crimeware-rootkit, November 2010.

[3] N. Daswani and M. Stoppelman. The Anatomy of Clickbot.A. In *Proc. HotBots*, 2007.

[4] V. Dave, S. Guha, and Y. Zhang. Measuring and Fingerprinting Click-spam in Ad Networks. In *Proceedings of ACM SIGCOMM*, 2012.

[5] M. Giuliani. ZeroAccess, an advanced kernel mode rootkit. http://www.prevx.com/blog/171/ZeroAccess-an-advanced-kernel-mode-rootkit.html, April 2011.

[6] Google Ads: Ad Traffic Quality Resource Center Overview. http://www.google.com/ads/adtrafficquality/.

[7] Google Inc. About smart pricing. *AdWords Help*, Apr. 2013. http://support.google.com/adwords/answer/2604607.

[8] F. Howard. Exploring the Blackhole exploit kit. http://nakedsecurity.sophos.com/exploring-the-blackhole-exploit-kit/.

[9] L. Kim. The most expensive keywords in Google AdWords. http://www.wordstream.com/blog/ws/2011/07/18/most-expensive-google-adwords-keywords, July 2011.

[10] B. Krebs. Takedowns: The Shuns and Stuns That Take the Fight to the Enemy. In *McAfee Security Journal*, volume 6, 2010.

[11] B. Krebs. Fake Antivirus Industry Down, But Not Out. http://krebsonsecurity.com/2011/08/fake-antivirus-industry-down-but-not-out/, August 2011.

[12] C. Kreibich, N. Weaver, C. Kanich, W. Cui, and V. Paxson. GQ: Practical Containment for Measuring Modern Malware Systems. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 397–412. ACM, 2011.

[13] The Lote clicking agent. http://www.clickingagent.com/.

[14] C. Grier et al. Manufacturing Compromise: The Emergence of Exploit-as-a-Service. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, October 2012.

[15] K. McNamee. Malware Analysis Report. Botnet: ZeroAccess/Sirefef. http://www.kindsight.net/sites/default/files/Kindsight_Malware_Analysis-ZeroAcess-Botnet-final.pdf, February 2012.

[16] B. Miller, P. Pearce, C. Grier, C. Kreibich, and V. Paxson. What's Clicking What? Techniques and Innovations of Today's Clickbots. In *Proc. Detection of Intrusions and Malware & Vulnerability Assessment*, 2011.

[17] A. Neville and R. Gibb. ZeroAccess Indepth (Symantec Corporation White Paper). http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/zeroaccess_indepth.pdf, October 2013.

[18] Pricewaterhouse Coopers. IAB Internet Advertising Revenue Report: 2013 First Six Months' Results. http://www.iab.net/media/file/IAB_Internet_Advertising_Revenue_Report_HY_2013.pdf, October 2013.

[19] E. Rodionov and A. Matrosov. The Evolution of TDL: Conquering x64. http://www.eset.com/us/resources/white-papers/The_Evolution_of_TDL.pdf.

[20] L. Sinclair. Click fraud rampant in online ads, says Bing. http://www.theaustralian.com.au/media/click-fraud-rampant-in-online-ads-says-bing/story-e6frg996-1226056349034, 2011.

[21] A. Tuzhilin. The Lanes Gifts v. Google Report. http://googleblog.blogspot.com/pdf/Tuzhilin_Report.pdf, 2005.

[22] J. Wyke. The ZeroAccess Botnet: Mining and Fraud for Massive Financial Gain. http://www.sophos.com/en-us/why-sophos/our-people/technical-papers/zeroaccess-botnet.aspx, September 2012.

[23] J. Wyke. ZeroAccess. http://www.sophos.com/en-us/why-sophos/our-people/technical-papers/zeroaccess.aspx, 2012.

A ZeroAccess Algorithms

A.1 ZeroAccess Search-Hijacking Ad List Decryption Algorithm

The SH-C&C server encrypts its responses to requests by the search-hijacking module for replacement ad URLs. The following code listing gives the decryption algorithm for this response. We extracted the code from the binary module and decompiled it using the IDA Pro Hexrays decompiler.

```
// Usage:
//    char cipher_text[4096];
//    uint32_t cipher_len = fread(cipher_text, .. );
//    char cipher_len_str[256];
//    uint32_t strlen_cipher_len_str = sprintf(cipher_len_str, "%u", cipher_len);
//    decrypt (
//            cipher_len_str,
//            cipher_text,
//            cipher_len,
//            strlen_cipher_len_str
//    );

int decrypt (
  char* cipher_len_str,
  char* cipher_text,
  uint32_t cipher_len,
  uint32_t strlen_cipher_len_str)
{
  char* v4;
  uint32_t v5;
  int32_t v6, v7, result;
  char v9;
  int32_t v10;
  char v11;
  int32_t v12, v13, v14;
  char v15;
  int32_t v16;
  char v17;
  int32_t v18;
  uint8_t v19;
  int32_t v20;
  uint32_t v21;
  uint8_t v22;
  char v23;
  char *v24;
  char v25, v26;
  int32_t v27;
  char arr[256];

  v4 = cipher_len_str;
  v26 = 0;
  v5 = 0;
  do {
    *(arr + v5) = (char)v5;
    ++v5;
  } while (v5 < 256);

  v6 = 0;
  v7 = 0;
  v27 = 0;
  result = 0;
  while (1) {
    v9 = *(arr + v6);
    if (result >= strlen_cipher_len_str)
      result = 0;

    v10 = (uint8_t)(v7 + v9 + *(char *)(result + v4));

    *(arr + v6) = *(arr + v10);
    *(arr + v10) = v9;

    v11 = *(arr +1 + v27);
    v12 = result + 1;


    if (v12 >= strlen_cipher_len_str)
      v12 = 0;

    v13 = (uint8_t)(v10 + v11 + *(char *)(v12 + v4));

    *(arr + 1 + v27) = *(arr + v13);
    v14 = v12 + 1;
    *(arr + v13) = v11;

    v15 = *(arr+2 + v27);
    if (v14 >= strlen_cipher_len_str)
      v14 = 0;

    v16 = (uint8_t)(v13 + v15 + *(char *)(v14 + v4));
    *(arr+2 + v27) = *(arr + v16);
    *(arr + v16) = v15;

    v17 = arr[v27+3];
    v18 = v14 + 1;
    if (v18 >= strlen_cipher_len_str)
      v18 = 0;

    v19 = v16 + v17 + *(char *)(v18 + v4);
    v20 = v27;
    v7 = v19;

    arr[v27+3] = *(arr + v7);
    *(arr + v7) = v17;
    result = v18 + 1;
    v27 += 4;

    if ((unsigned int)(v20 + 4) >= 0x100)
      break;
    v6 = v27;
  }

  v21 = 0;
  if (cipher_len) {
    v22 = 0;
    do {
      ++v22;
      v23 = *(arr + v22);
      v24 = arr + (uint8_t)(v23 + v26);
      v26 += v23;
      v25 = *v24;
      *(arr + v22) = *v24;
      *v24 = v23;
      result = (uint8_t)(v25 + v23);
      *(char *)(v21++ + cipher_text) ^= *(arr + result);
    } while (v21 < cipher_len);
  }

  return result;
}
```

A.2 ZeroAccess Search-Hijacking Pseudo-domain to IP Algorithm

The following Python fragment gives an algorithm for decoding IP addresses in the search-hijacking module represented as (pseudo-)domain names. We developed the code based on decompiling the binary.

```
from binascii import crc32
from struct import pack
from socket import inet_ntoa

def deobfuscate_domain(d):

    b0 = crc32(d[0:5],0x7E873D53) & 0xFF
    b1 = crc32(d[5:9],0x570848EB) & 0xFF
    b2 = crc32(d[9:12],0x768772F3) & 0xFF
    b3 = crc32(d[12:17],0x4775114F) & 0xFF

    ip_as_int = b0 + (b1 << 8)  \
                   + (b2 << 16) \
                   + (b3 << 24)
    packed_ip = pack(’<I’, ip_as_int)

    return inet_ntoa(packed_ip)
```